

REMARKS/ARGUMENTS

This Amendment and the following remarks are intended to fully respond to the Final Office Action dated August 26, 2004. In that Office Action, claims 1-34 and 38-58 were examined, and all claims were rejected. More specifically, claim 31 was objected to because of an informality; claims 1-4, 6, 7, 9-13, 15, 17-23, 25-34, 38-41, 43, 44, 46-50, 52 and 54-58 stand rejected under 35 U.S.C. § 102(e) as being anticipated by Johnston (USPN 6,189,142); and claims 5, 8, 14, 16, 24, 42, 45, 51 and 53 stand rejected under 35 U.S.C. § 103(a) as being unpatentable over Johnston, in view of Levine (USPN 6,349,406). Reconsideration of these objections and rejections, as they might apply to the original and amended claims in view of these remarks, is respectfully requested.

In this Response, no claims have been amended. No new claims have been added, and no claims have been canceled.

Claim Objections - Informality

Claim 31 was objected to because of informality. More specifically, Claim 31 was labeled as “original” when it was, in fact, amended. The label has been corrected to remove the informality, and applicant respectfully requests that the objection be withdrawn.

Claim Rejections - 35 U.S.C. § 102

Claims 1-4, 6, 7, 9-13, 15, 17-23, 25-34, 38-41, 43, 44, 46-50, 52 and 54-58 stand rejected under 35 U.S.C. § 102(e) as being anticipated by Johnston (USPN 6,189,142). Applicant respectfully traverses the Examiner’s rejections under 35 U.S.C. § 102(e), on the grounds that Johnston does not anticipate the present invention.

Before discussing the rejection of the claims, brief descriptions of Johnston and the present invention are provided.

Johnston relates to “providing runtime performance analysis in a visual programming environment.” (col. 1, lines 18-19) Under Johnston, “providing runtime performance analysis in a computing system having a capability for visual programming, compris[es] instrumenting a selected one of one or more visually-created programs according to a visual programming performance data collection technique, and gathering execution information using the instrumented program.” This instrumentation is achieved by “adding code hooks in appropriate

locations within the code that implements the visual program.” (col. 9, lines 49-51) That is, Johnston teaches adding code hooks for debugging or tracing the execution of the visually created program.

Code hooks are added to code by “adding a small piece of code in the appropriate place for each element of interest. In an object-oriented implementation, this comprises adding a message, or method invocation. In a procedural programming implementation, this comprises adding a function or subroutine call.” (col. 9, lines 52-59) “For purposes of the present invention, the invoked code logs the fact that the invoking element has executed; causes recording of execution timing information; and for attribute-to-attribute connections, logs the amount of data being moved.” (col. 9, lines 63-67)

As defined on page 10 of the description of the pending application, a history operand in source code represents a sequence of data associated with the history of an operand instance. A history operator in source code is a shorthand representation of a function that object code will perform on the history data represented by the history operand. That is, history operators and operands are special, syntactical constructs that are recognized by the compiler or interpreter of the source code and trigger the compiler or interpreter to generate a particular object code. History operators and operands are unique in that their values can be accessed by the source code, and can be used to determine code flow. Likewise, history operators and operands may be used to optimize and condense source code by substituting history operators and operands for traditional source code bookkeeping functionality. Examples of history operators and history operands, and of source code optimizations they enable, are shown in Figs. 4-12 of the pending application.

With these differences in mind, it will become clear as discussed below how the claims differ from the Johnston reference.

Under 35 U.S.C. § 102, a reference must show or describe each and every element claimed in order to anticipate the claims. *Verdegaal Bros. V. Union Oil Co. of California* 814 F.2d 628 (Fed. Cir. 1987) (“A claim is anticipated only if each and every element as set forth in the claim is found, either expressly or inherently described, in a single prior art reference.”). Since Johnston does not disclose recognizing a history operator and a history operand in the source code, Johnston cannot, as a matter of law, anticipate claim 1.

In the current office action, examiner has argued that “recognizing a history operator and a history operand in the source code” is taught by Johnston. In support, examiner went on to cite several sections of Johnston that describe code hooks. However, as explained above, history operators and operands are not the same thing as code hooks. History operators and operands have additional functionality that the code hooks described by Johnston lack. Namely, history operators and operands can be directly referenced by source code so that their resulting values can determine the code flow of a program, whereas code hooks merely run within the program and monitor code flow and execution without altering code flow beyond the existence of said monitoring. Further, history operators and operands can be used to optimize and condense source code, whereas code hooks, which do not return any value and so cannot be directly referenced by the source code, cannot be substituted to optimize and condense source code. For this and other reasons, applicant maintains that Johnston cannot anticipate recognizing a history operator and a history operand in the source code as in claim 1. Reconsideration is thus respectfully requested.

Since claims 2-4, 6-7, 9-13, 15, and 17-18 depend directly or indirectly from claim 1 such that those claims should also be allowed over Johnston, reconsideration is respectfully requested.

Likewise, Johnston does not disclose a history operand to direct a translator to generate first object code and a history operator to direct the translator to generate second object code, and so Johnston cannot, as a matter of law, anticipate claim 19. As explained previously, code hooks, as defined in Johnston, serve only to log and record data. Code hooks do not return values, and thus cannot be referred to by source code like history operands and operators. Since claims 20-23 and 25 depend directly or indirectly from claim 19 such that those claims should also be allowed over Johnston, reconsideration is respectfully requested.

Likewise, Johnston does not disclose recognizing a history operand in source code, and finding a least one instance of the history operand in the source code in response to recognizing the history operand, and so Johnston cannot, as a matter of law, anticipate claim 26. As explained previously, code hooks, as defined in Johnston, serve only to log and record data. Code hooks do not return values, and thus cannot be referred to by source code like history operands. Since claims 27-30 depend directly or indirectly from claim 26 such that those claims should also be allowed over Johnston, reconsideration is respectfully requested.

Likewise, Johnston does not disclose recognizing a history operand in source code, and finding a least one instance of the history operand in the source code, and so Johnston cannot, as a matter of law, anticipate claim 31. As explained previously, code hooks, as defined in Johnston, serve only to log and record data. Code hooks do not return values, and thus cannot be referred to by source code like history operands. Since claims 32-34 depend directly or indirectly from claim 31 such that those claims should also be allowed over Johnston, reconsideration is respectfully requested.

Likewise, Johnston does not disclose recognizing a history operator and a history operand in the source code, and performing the history operator on a data history, and so Johnston cannot, as a matter of law, anticipate claim 38. As explained previously, code hooks, as defined in Johnston, serve only to log and record data. Code hooks do not return values, and thus cannot be referred to by source code like history operands and operators. Since claims 39-41, 43-44, 46-50, 52, and 54 depend directly or indirectly from claim 38 such that those claims should also be allowed over Johnston, reconsideration is respectfully requested.

Likewise, Johnston does not disclose recognizing a history operand in source code, and finding at least once instance of the history operand in the source code in response to recognizing the history operand, and so Johnston cannot, as a matter of law, anticipate claim 55. As explained previously, code hooks, as defined in Johnston, serve only to log and record data. Code hooks do not return values, and thus cannot be referred to by source code like history operands. Since claims 56-58 depend directly or indirectly from claim 55 such that those claims should also be allowed over Johnston, reconsideration is respectfully requested.

Claim Rejections - 35 U.S.C. § 103

Claims 5, 8, 14, 16, 24, 42, 45, 51 and 53 stand rejected under 35 U.S.C. § 103(a) as being unpatentable over Johnston, in view of Levine (USPN 6,349,406). Applicant respectfully traverses the § 103 rejections, as the Examiner has failed to establish a prima facie case of obviousness. In order to establish a prima facie case, the references must show that the cited references teach or suggest each of the elements of the claimed invention (MPEP § 706.02(j) and 2142-43). With respect to the cited references, all elements of the independent claims are neither taught nor suggested.

Johnston, as discussed above, relates to “providing runtime performance analysis in a visual programming environment.” (col. 1, lines 18-19) Under Johnston, “providing runtime performance analysis in a computing system having a capability for visual programming, compris[es] instrumenting a selected one of one or more visually-created programs according to a visual programming performance data collection technique, and gathering execution information using the instrumented program.” This instrumentation is achieved by “adding code hooks in appropriate locations within the code that implements the visual program.” (col. 9, lines 49-51) That is, Johnston teaches adding code hooks for debugging or tracing the execution of the visually created program.

Code hooks are added to code by “adding a small piece of code in the appropriate place for each element of interest. In an object-oriented implementation, this comprises adding a message, or method invocation. In a procedural programming implementation, this comprises adding a function or subroutine call.” (col. 9, lines 52-59) “For purposes of the present invention, the invoked code logs the fact that the invoking element has executed; causes recording of execution timing information; and for attribute-to-attribute connections, logs the amount of data being moved.” (col. 9, lines 63-67)

In contrast, Levine relates to “a method and system for compensating for instrumentation overhead in trace data by computing average minimum event times.” (Abstract) More specifically, under Levine “in order to profile a program, the program is executed to generate trace records that are written to a trace file.” (Col. 3, lines 16-18) The “trace data may be generated via selected events and timers through the instrumented interpreter without modifying the source code.” (Col. 8, lines 14-16) The interpreter executes a trace program “used to record data upon the execution of a hook, which is a specialized piece of code at a specific location in a routine or program in which other routines may be connected.” (Col. 9, lines 43-46) However, Levine does not teach or suggest using a history operator and a history operand as described in the pending application, nor do Levine’s hooks return data that can be accessed in source code and used to make code flow determinations. Rather, Levine, similar to Johnston, teaches trace data generated by an interpreter upon execution of a hook code in the source program.

The combination of Johnston and Levine simply does not teach or suggest each of the elements of the claimed invention. Neither Johnston nor Levine, alone or in combination, teach

or suggest recognizing a history operator and a history operand in the source code as recited in claim 1 and thus incorporated in dependent claims 5, 8, 14, and 16. Reconsideration of the § 103(a) rejections is therefore respectfully requested.

Likewise, the combination of Johnston and Levine does not teach or suggest a history operand to direct a translator to generate first object code and a history operator to direct the translator to generate second object code as recited in claim 19 and thus incorporated in dependent claim 24. Reconsideration of the § 103(a) rejections is therefore respectfully requested.

Likewise, the combination of Johnston and Levine does not teach or suggest recognizing a history operator and a history operand in the source code, and performing the history operator on a data history as recited in claim 38 and thus incorporated in dependent claims 42, 45, 51, and 53. Reconsideration of the § 103(a) rejections is therefore respectfully requested.

Since the remarks above are believed to distinguish over the applied reference, any remaining arguments supporting the claim rejections are not acquiesced to because they are not addressed herein.

Conclusion

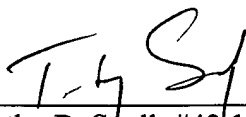
It is believed that no further fees are due with this Response. However, the Commissioner is hereby authorized to charge any deficiencies or credit any overpayment with respect to this patent application to deposit account number 13-2725.

In light of the above remarks and amendments, it is believed that the application is now in condition for allowance, and such action is respectfully requested. Should any additional issues need to be resolved, the Examiner is requested to telephone the undersigned to attempt to resolve those issues.

Respectfully submitted,

Dated: 10/14/04





Timothy B. Scull, #42,137
MERCHANT & GOULD P.C.
P.O. Box 2903
Minneapolis, MN 55402-0903
303.357.1648